Intuitionistic Logic

Tutorial – CADE-27

Giselle Reis – giselle@cmu.edu
Carnegie Mellon University, Qatar

August 23, 2019

1 What is logic?

The history of logic goes as far back as the Greek philosophers. In fact, the mastery of logic was considered essential before studying other topics (unfortunately this is no longer the case nowadays). But in those times there were no formulas or connectives, it was all done using natural language. Nevertheless, the main goal was the same: to arrive at *conclusions* starting from a valid set of *premises* and using sound *inferences*.

In the very beginning (i.e. 500 B.C. to 19^{th} century), the derivation of conclusions was done via *arguments* between two parties: proponent and opponent. They would agree on an initial set of premises and the proponent would put forward a fact that necessarily follows from them. The opponent's role is to check if this is indeed the case, and provide a counter-example if it is not. By iterating this process several times, the proponent and opponent can agree on the overall conclusion. One can say that the premises and agreed reasoning steps consists (loosely) of a *proof* of that conclusion.

Since then, a lot has changed. Natural language was abandoned and a symbolic system, with fewer ambiguities, was developed. Logic began to be used more and more for mathematical reasoning[1]. Systems for manipulating the symbols and developing proofs were invented (*proof systems*). New symbols or new interpretations of the classical symbols gave rise to new logics, which served to model other kinds of reasoning (from philosophy to computer science).

In this tutorial we will be concerned with one of these logics that deviated from the "classical" interpretation of connectives and is very relevant for computer science.

[1] This part of logic's history is nicely described in a comic book called *Logicomix*

1.1 Why another logic?

There were (and still are) many discussions on what constitutes "correct" reasoning, which steps one can take without compromising an argument, and what it means for something to be true. But one thing that people usually accept fairly naturally is that, for every proposition A, either A holds or $\neg A$ holds. This is the so-called *law of excluded middle*. Given this principle, a proof that it is not the case that $\neg A$ (i.e., a proof of $\neg\neg A$) can be considered as evidence for A (if the disjunction is true and we know that one disjunct does not hold, then the other one **must** be true). A proof that relies heavily on the law of excluded middle is the following.

Theorem 1. *There exist two irrational numbers x and y such that x^y is a rational number.*

Proof. Take the number $\sqrt{2}^{\sqrt{2}}$. We do not know if this is a rational or irrational number, but the law of excluded middle tells us it must be one or the other.
Case 1: $\sqrt{2}^{\sqrt{2}}$ is rational. Then choose $x = y = \sqrt{2}$ and the theorem holds.
Case 2: $\sqrt{2}^{\sqrt{2}}$ is irrational. Then choose $x = \sqrt{2}^{\sqrt{2}}$ and $y = \sqrt{2}$. Therefore $x^y = 2$ and the theorem also holds. □

Is this a valid proof of the theorem or not? One might argue that it is a proof, although not a very informative one. Some were not happy about this situation, and they decided to come up with new rules for the game. They were the constructivists (or intuitionists). They decided that, in their logic, the truth of a judgment is solely determined by an evidence (or proof) of **that** judgment. Not a negation of its negation, but the judgment. It can be thought of as a *proof-centered* logic. It turns out that proofs become really interesting and informative, and can even be interpreted as algorithms (spoiler alert!). Mathematically, intuitionistic proofs represent the construction of objects (hence the name)[2]. A real intuitionistic proof of the theorem above would actually show how to obtain values for x and y which satisfy the property.

1.2 Preliminaries

We will work with propositional logic (i.e. no quantifiers). In this section, we define the syntax and some conventions used.

The operators of propositional logic are:

- conjunction (i.e. *and*): $\wedge$
- disjunction (i.e. *or*): $\vee$

[2] Actually, there is a whole field named *constructive mathematics* trying to formalize all mathematics in terms of intuitionistic (constructive) proofs.

- implication: $\supset$
- false: $\bot$
- true: $\top$

Note that we do not use negation ($\neg$). Instead, a formula $\neg A$ will be written as $A \supset \bot$. We use upper case letters for *formula variables*, i.e., placeholders which can be replaced by an arbitrary formula (like the A used before). Many times we develop proofs using formula variables, since it results in a more general proof. Lower case letters are reserved for propositions.

It is important to stress at this point that the notation conventions used here are not set in stone and may be used differently in different sources. After getting used with logic, it may be easy to deduce from the context what should be a formula and what should be formula variables, but to avoid confusion, we will try to follow these conventions closely.

Definition 1 (Formulas)**.** *The set of* formulas *is inductively defined as follows:*

- *A propositional variable (or constant) $p, q, r, s...$ is an (atomic) formula.*
- *$\top$ and $\bot$ are formulas.*
- *If A and B are formulas, then*
  - *$A \wedge B$ is a formula.*
  - *$A \vee B$ is a formula.*
  - *$A \supset B$ is a formula.*

To avoid the use of parenthesis, we adopt the following convention on the precedence of logical operators (the ones listed first are evaluated first): $\wedge, \vee, \supset$. Therefore $p \wedge q \vee r$ is equivalent to $(p \wedge q) \vee r$, for example. Also, $\supset$ associates to the right: $p \supset q \supset r$ is equivalent to $p \supset (q \supset r)$.

Example 1. *The following* ***formula*** *represents the fact that being human implies being mortal:*

$$\mathsf{human} \supset \mathsf{mortal}$$

The names human *and* mortal *are* ***propositional constants****.*

As another example, the following formula encodes that fact that if an animal has feathers, a beak, and lays eggs, it is a bird.

$$\mathsf{feathers} \wedge \mathsf{beak} \wedge \mathsf{eggs} \supset \mathsf{bird}$$

In order to build proofs we need to use a *proof calculus*. There are many different ones, using the most various notations. In this tutorial we will see two of them: natural deduction and sequent calculus. Both use the same kind of building blocks: *inference rules*. They look like this:

$$\frac{\square \quad \cdots \quad \square}{\square}\ name$$

The red boxes are placeholders for judgments called *premises* and the blue box for a judgment called *conclusion*. A judgment is simply an assessment about an object and it is written as *object judgment* (e.g. we can judge if a string of digits, dots (.) and commas (,) is a valid number: 42 *number* is a judgment that holds whereas $1.42, 5.2$ *number* does not).

An inference rule is interpreted as: if all the premises hold, then the conclusion holds. This means that truth is preserved from the top down. Each inference rule has a name, which is noted on the right side of the inference line.

Inference rules are presented with *schema variables*, represented by uppercase letters (e.g., if the judgments involve formulas, then formula variables are used). These variables can be replaced by arbitrary objects and no matter what you use for the variable, the rule will still be correct! For example, suppose we are working with judgments of the type N *nat*, i.e., N is a natural number. We can have an inference rule called $+$:

$$\frac{N\ nat \quad M\ nat}{N + M\ nat}\ +$$

When using this rule, it does not matter if we have $N = 1$ and $M = 3$ or $N = 78918749$ and $M = 234 * 200 + 789 * 90$, the inference is correct regardless. Even if a crazy person decides to use "a" *nat* as one of the premises, the rule can still be applied. One premise will be false, and can never be proved, but that does not invalidate the fact that adding a natural number to another natural number results in a natural number.

An inference rule can also have zero premises. This means that the judgment in the conclusion holds independently of premises. This might be because the judgment is trivial (e.g. $x = x$), it is an axiom, or because it is one of the assumptions we made in the proof.

A proof is constructed by plugging in inference rules together. It can be represented as a tree (upside-down considering how trees are drawn in computer science; but in the right direction considering real trees) or a DAG (directed acyclic graph) and it is considered complete (or closed) only when all the leaves are inferences with no premises. Proofs with open leaves are sometimes called *derivations* to disambiguate. The root of the tree is the judgment being proved.

2 Natural Deduction

David Hilbert was a German mathematician who published, in 1899, a very influential book called *Foundations of Geometry*. The thing that was so revolutionary about this book was the way theorems were proven. It followed very closely an axiomatic approach, meaning that a set of geometry axioms were assumed and the whole theory was developed by deriving new facts from those axioms via *modus ponens* basically[3]. Hilbert believed this was the right way to do mathematics, so in 1920 he proposed that all mathematics be formalized using a set of axioms, which needed to be proved consistent (i.e. $\bot$ cannot be derived from them). All further mathematical knowledge would follow from this set.

Shortly after that, it was clear that the task would not be so easy. Several attempts were proposed, but mathematicians and philosophers alike kept running into paradoxes. The foundation of mathematics was in crisis: to have something so fundamental and powerful be inconsistent seemed like a disaster. Many scientists were compelled to prove consistency of at least some part of math, and this was the case for Gentzen (another German mathematician and logician). In 1932 he set as a personal goal to show consistency of logical deduction in arithmetic. Given a formal system of logical deduction (i.e., a *proof system*), proving its consistency becomes a purely mathematical problem, so that is what he did first. He thought that, later on, arithmetic could be added (turns out that this was not so straightforward, and he ended up developing yet another system, but ultimately Gentzen did prove consistency of arithmetic). The formal system of logical deduction developed was *natural deduction*, presented in a paper from 1934. In his words (translated to English): "First I wished to construct a formalism that comes as close as possible to actual reasoning. Thus arose a *calculus of natural deduction*."

2.1 Inference rules

We have said that intuitionistic logic is all about proofs, but what does it mean? It means that, in order to decide if a judgement of the type A *true* holds (meaning that the formula A is true), we need a proof of it. Since A is a logical formula (or proposition), the existence of a proof will be enforced by the way the logical connectives are defined. We start with $\wedge$.

Conjunction

$A \wedge B$ has a proof iff A has a proof and B has a proof[4].

[3] Nowadays, such reasoning systems are known as Hilbert calculi.

[4] *iff* = if and only if = $\Leftrightarrow$

It may look a bit silly, but defining the connectives in terms of proofs, as opposed to truth values, makes a big difference. It even results in a different logic.

Notice that this statement has two directions. We will use both of them to come up with valid inference rules. The backward direction tells us that if A has a proof and B has a proof (these are the premises), we can conclude that $A \wedge B$ has a proof. This fact can be represented by be following inference rule:

$$\frac{A \textit{ true} \quad B \textit{ true}}{A \wedge B \textit{ true}} \wedge I$$

The rule is named $\wedge I$ ("and introduction") because it is introducing the connective $\wedge$ between two propositions given the assumptions that there are proofs for them. The connective is also "introduced" when we read the definition from right to left.

The forward direction of the definition tells us that if $A \wedge B$ has a proof, we can conclude two things: (1) A has a proof; and (2) B has a proof. This is represented thus by two inference rules:

$$\frac{A \wedge B \textit{ true}}{A \textit{ true}} \wedge E_1 \qquad \frac{A \wedge B \textit{ true}}{B \textit{ true}} \wedge E_2$$

These rules are named $\wedge E_i$ ("and elimination") because they are eliminating a logical connective to reach a conclusion. The connective is also eliminated when reading the definition from left to right.

Implication

The case of implication is more interesting. This is the definition of provability of a formula with implication:

> $A \supset B$ has a proof iff the existence of a proof of A implies B has a proof.

Reading from right to left, we have that if the existence of a proof of A implies a proof of B, then we can conclude $A \supset B$. To represent "the existence of a proof of A", we will use an inference without premises: $\dfrac{}{A \textit{ true}}$. It is part of our assumptions that A has a proof no matter what. This assumption will use a label that will be the same as the one used in the name of the rule. This is used to keep track of where assumptions are coming from. Then, using the assumption zero or more times, we need to get a proof of B *somehow*. This *somehow* part is represented by the three

dots in the inference rule, which has to be filled in to complete the proof. Finally, we can conclude $A \supset B$.

$$\dfrac{\dfrac{}{A\ true}\,u \\ \vdots \\ B\ true}{A \supset B\ true}\ \supset I^u$$

Analogous to the $\wedge$ case, this rule is called $\supset I^u$ ("implication introduction"). It is important to note that the assumption A *true* can only be used in this part of the proof, i.e., to prove B *true*, and not anywhere else.

Looking at the definition from left to right we have that if $A \supset B$ has a proof then we can conclude B, but only if A itself has a proof. In order to get the conclusion, we actually have two conditions, that are translated as two premises.

$$\dfrac{A \supset B\ true \quad A\ true}{B\ true}\ \supset E$$

This rule is called, unsurprisingly, $\supset E$ ("implication elimination").

Truth

We look now at the case for $\top$. This one is easy, $\top$ is true by definition and this does not require a proof, so we have a rule without premises:

$$\dfrac{}{\top\ true}\ \top I$$

On the other hand, given $\top$ we cannot conclude anything interesting, so there is no elimination rule for it.

Disjunction

The case for $\vee$ is a bit more complicated. The provability of a disjunctive formulas is defined as:

> $A \vee B$ has a proof iff A has a proof or B has a proof.

The right to left reading translates straightforwardly to rules: if we have a proof of A, we also have a proof of $A \vee B$; analogous for B.

$$\dfrac{A\ true}{A \vee B\ true}\ \vee I_1 \qquad \dfrac{B\ true}{A \vee B\ true}\ \vee I_2$$

Now, what can we conclude from a proof of $A \vee B$? Certainly not A (what if the provable disjunct was B?), and symmetrically, not B. Is disjunction a degenerate connective that has only introduction rules? No. There must be something we can get from $A \vee B$. Let's call this something C. When can we conclude a new (possibly completely unrelated) proposition C from a proof of $A \vee B$? Well, we know that it must be the case that either A or B have a proof (reading the definition from left to right), but we do not know exactly which one. If, under the assumption of A, we can conclude C, and also under the assumption of B we conclude C, then we can definitely get C from a proof of $A \vee B$. Note that it is independent of which disjunct was actually true, since we have both cases covered!

How is this translated to a rule? If we look at it carefully, we are concluding C under three premises: (1) $A \vee B$ has a proof; (2) assuming A we get a proof of C; and (3) assuming B we also get a proof of C. Therefore:

$$\dfrac{A \vee B \text{ true} \quad \begin{array}{c}\overline{A \text{ true}}\,u \\ \vdots \\ C \text{ true}\end{array} \quad \begin{array}{c}\overline{B \text{ true}}\,v \\ \vdots \\ C \text{ true}\end{array}}{C \text{ true}} \vee E^{u,v}$$

This is a tricky one, so take your time to absorb it and read it again if you need. If you are wondering if this rule is really necessary, a convincing argument is that commutativity of disjunction ($(A \vee B) \supset (B \vee A)$) would not be provable without it.

Falsehood

The last propositional connective is $\bot$, which is simply falsehood, so we should not be able to prove it (if you do, rest assured that there is something wrong!). Therefore, there is no introduction rule. On the other hand, if we happen to derive $\bot$ in the middle of a proof, then we can conclude basically anything:

$$\dfrac{\bot \text{ true}}{C \text{ true}} \bot E$$

The set of all these rules together make up the *natural deduction* calculus for intuitionistic logic (Figure 1).

Notice that simply writing somethings above a line and another thing below does not make an inference rule. They are inspired by the meaning of the connectives, i.e. what it means for a proposition to be true. On the next lecture we will see how to formally prove that they are "just enough", and that the resulting system is well-behaved.

$$\frac{A\ true \quad B\ true}{A \wedge B\ true}\ \wedge I \qquad \frac{A \wedge B\ true}{A\ true}\ \wedge E_1 \qquad \frac{A \wedge B\ true}{B\ true}\ \wedge E_2$$

$$\frac{\begin{array}{c}\overline{}\ u\\ \vdots\\ B\ true\end{array}}{A \supset B\ true}\ \supset I^u \qquad \frac{A \supset B\ true \quad A\ true}{B\ true}\ \supset E$$

$$\frac{A\ true}{A \vee B\ true}\ \vee I_1 \qquad \frac{B\ true}{A \vee B\ true}\ \vee I_2 \qquad \frac{A \vee B\ true \quad \begin{array}{c}\overline{}\ u\\ \vdots\\ C\ true\end{array} \quad \begin{array}{c}\overline{}\ v\\ \vdots\\ C\ true\end{array}}{C\ true}\ \vee E^{u,v}$$

$$\frac{\bot\ true}{C\ true}\ \bot E \qquad \frac{}{\top\ true}\ \top I$$

Figure 1: Natural deduction system for intuitionistic logic (NJ)

2.2 Example

As an example of a proof in Natural Deduction, take the proof of $A \wedge (A \wedge A \supset B) \supset B$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \wedge (A \wedge A \supset B)\ true}\ u}{(A \wedge A) \supset B\ \ true}\ \wedge E_2 \quad \dfrac{\dfrac{}{A\ \ true}\ w \quad \dfrac{}{A\ \ true}\ w}{A \wedge A\ \ true}\ \wedge I}{B\ \ true}\ \supset E}{A \supset B\ \ true}\ \supset I^w \quad \dfrac{\dfrac{}{A \wedge (A \wedge A \supset B)\ true}\ u}{A\ \ true}\ \wedge E_1}{B\ \ true}\ \supset E$$
$$\frac{}{A \wedge (A \wedge A \supset B) \supset B\ \ true}\ \supset I^u$$

You may notice that this proof has a few redundant steps. This is not by accident. It could be simplified, but it could also be more redundant, as we will see in the next part.

3 Curry-Howard Isomorphism

The Curry-Howard correspondence is not a thing that was suddenly discovered, formalized and given a name. It is actually the organization of several observations made trough many years by different people. Little by little people were realizing that those observations were actually the same, and then they decided to make it a thing. As it is known today, the Curry-Howard correspondence establishes a relation between formulas and proofs of those formulas in propositional intuitionistic logic and functions of a given type in a functional programming language. More concretely, if we interpret atomic propositions as basic types and logical connectives as type constructors (in OCaml: `*`, `->`, `|`), then the proof of a formula corresponds to a program of the associated type.

First of all, let us see how formulas can be viewed as types.

The smallest piece of a formula is an atomic proposition, and the smallest piece of a type is an atomic type, so it is only natural that we relate one to the other. Since we usually work with proofs using formula variables A, B, C, we will consider only type variables.

Conjunction translates as the product type: $A \wedge B$ assumes proofs of both A and B, analogously, `'a * 'b` assumes two terms (or programs, or functions, however you want to call them), one of type `'a` and the other of type `'b`.

Implication is straightforwardly translated to the function type: a proof of $A \supset B$ goes from some given assumption A to B, the same way that a function `'a -> 'b` takes an argument of type `'a` and computes a value of type `'b`.

Disjunction represents a choice, which is incorporated by the union type: an option type is either `Some` or `None`, a list is either `[]` (empty) or `x::L` (cons of at least one element), etc.

True is simply a fact that carries no information, so it is interpreted as the unit type (the type with one inhabitant).

Finally, false corresponds to the empty type (or bottom type), a type that has no values (and usually not available in programming languages). When it exists, the empty type is used to signal functions that do not return anything (e.g. raising exceptions or not terminating).

These relations are summarized in Table 1.

3.1 Proof terms

Now that we know that formulas can be types, we will redesign natural deduction to annotate formulas with terms (i.e., expressions) of the corresponding type. In particular we will use the new judgment:

[5] Not available in OCaml. Available in Scala as `Nothing` and in Rust as `never`.

Logic world	**Programming world** (OCaml)
formula variables (A, B, C, etc)	type variables (`'a`, `'b`, `'c`, etc.)
conjunction ($\wedge$)	product type (`*`)
implication ($\supset$)	arrow type (`->`)
disjunction ($\vee$)	union type (`|`)
true ($\top$)	unit type (`unit`)
false ($\bot$)	empty type[5]

Table 1: Formulas as types

$$M : A$$

to denote that the term M has type A. Therefore, the right side of the colon contains the logical connectives and formulas we are familiar with. The rules on these formulas will be the same ones as we know. The left side of the colon contains a term (in a functional-like programming language) of type A. We will now construct these terms.

A good thing to keep in mind is the duality between introduction and elimination rules. Reading them top-down, introduction rules construct formulas from smaller pieces and elimination rules extract the pieces from the formulas. Analogously, introduction rules will construct terms and elimination rules will deconstruct them.

Conjunction

Conjunction is represented as the product type. So what term in programming would have type `'a * 'b`? A pair! So the introduction rule for conjunction is also a rule that takes two terms M and N, of types A and B, and puts them together in a pair structure.

$$\frac{M : A \quad N : B}{\langle M, N \rangle : A \wedge B} \wedge I$$

There are two elimination rules for conjunction, each conclude one side of the formula. If the term M is a pair of type $A \wedge B$, what is the operation performed on M to get the first element of the pair? What about to get the second? The pieces of a pair can be obtained by **fst** and **snd** operations, therefore, the conclusions of the elimination rules are exactly the application of these operations to term M.

$$\frac{M : A \wedge B}{\textbf{fst}\ M : A} \wedge E_1 \qquad \frac{M : A \wedge B}{\textbf{snd}\ M : B} \wedge E_2$$

Implication

Implication is the function type, so which term in programming has the function type? Well, a function! Since we do not care about function names, we will represent them anonymously using λ abstractions[6]. The introduction rule for implication takes an A and constructs B, which will be represented by a term, or function, that takes an argument u of type A and passes it to a term M of type B, i.e, $\lambda u.M$.

$$\dfrac{\begin{array}{c}\overline{u : A} \\ \vdots \\ M : B\end{array}}{\lambda u : A.\ M : A \supset B}\ \supset I^u$$

Implication elimination is quite intuitive. If you have a term of type $A \supset B$ (i.e., a function that takes something of type A as an argument and returns something of type B), and you have a term of type A, what do you do? Apply the function! So $\supset E$ is simply function application.

$$\dfrac{M : A \supset B \quad N : A}{MN : B}\ \supset E$$

Disjunction

Disjunction is the union type, which is represented in OCaml by the `|` used in `type`s. So if we have a term of type A, how do we construct a term of type $A \vee B$? Let's look at an example. Suppose we have a datatype for users in our system, and the users can be identified either by a name or by a user ID:

```
type user = Name of string | UserID of int
```

Now, someone is registering and provided a user name: `"aristotle"`. How do we construct a user from it? We use the type constructor and apply it to the string: `Name("aristotle")`. The operation of applying the type constructor to build a union type is called *injection*. Given a term M of type A, we will use **inl** to construct the union type $A \vee B$ and **inr** if M has type B. The injections will be annotated with the type of the other component in the union (although in programming languages this is completely omitted).

$$\dfrac{M : A}{\mathbf{inl}^B M : A \vee B}\ \vee I_1 \qquad \dfrac{M : B}{\mathbf{inr}^A M : A \vee B}\ \vee I_1$$

[6] You can think of a $\lambda x.$ abstraction as a `fun x ->` in OCaml.

The deconstruction of a union type is done by casing on the type constructors (the injections) and extracting the term inside it, so that's exactly what we will do for the disjunction elimination rule. We start with a term M of type $A \vee B$. The other two assumptions in the rule gives us ways to construct some term N or O (possibly different) of type C if we are given terms u and v of types A and B, respectively. When casing on M and deconstructing this union, we have a case for each possible term u or v that produces some term N or O, both of type C. Notice the consistency of the types in the case statement and the scope of the terms u and v.

$$\frac{\begin{array}{ccc} & \overline{u : A} & \overline{v : B} \\ & \vdots & \vdots \\ M : A \vee B & N : C & O : C \end{array}}{\textbf{case } M \textbf{ of inl } u \Rightarrow N \mid \textbf{inr } v \Rightarrow O : C} \vee E$$

True

True is the unit type, which has only one inhabitant. We will represent it by $\langle\rangle$ (the fact that this is an empty pair is not a coincidence ;) in the rule $\top I$. There is no deconstruction of unit, the same way that there is no elimination rule for $\top$.

$$\frac{}{\langle\rangle : \top} \top I$$

False

False represents the empty type, which has no inhabitants. So if we constructed a term M of type $\bot$, we can be sure that M is a nonsense program and we can abort it. An aborted program may have any type C, which will be used to annotate the command.

$$\frac{M : \bot}{\textbf{abort } M : C} \bot E$$

3.1.1 Example

All the rules from the previous section are summarized in Figure 2. Let's see them in action now by using the redundant proof of $A \wedge (A \wedge A \supset B) \supset B$:

$$\frac{M : A \quad N : B}{\langle M, N\rangle : A \wedge B}\ \wedge I \qquad \frac{M : A \wedge B}{\mathbf{fst}\ M : A}\ \wedge E_1 \qquad \frac{M : A \wedge B}{\mathbf{snd}\ M : B}\ \wedge E_2$$

$$\frac{\begin{array}{c}\overline{u : A}\\ \vdots\\ M : B\end{array}}{\lambda u : A.\ M : A \supset B}\ \supset I^u \qquad \frac{M : A \supset B \quad N : A}{MN : B}\ \supset E$$

$$\frac{M : A}{\mathbf{inl}^B M : A \vee B}\ \vee I_1 \qquad \frac{M : B}{\mathbf{inr}^A M : A \vee B}\ \vee I_1$$

$$\frac{\begin{array}{ccc} & \overline{u : A} & \overline{v : B}\\ & \vdots & \vdots\\ M : A \vee B & N : C & O : C\end{array}}{\mathbf{case}\ M\ \mathbf{of}\ \mathbf{inl}\ u \Rightarrow N \mid \mathbf{inr}\ v \Rightarrow O : C}\ \vee E$$

$$\frac{M : \bot}{\mathbf{abort}\ M : C}\ \bot E \qquad \frac{}{\langle\rangle : \top}\ \top I$$

Figure 2: Natural deduction annotated with proof terms.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \wedge (A \wedge A \supset B)\ true}\,u}{(A \wedge A) \supset B\ \ true}\ \wedge E_2 \quad \dfrac{\dfrac{}{A\ \ true}\,w \quad \dfrac{}{A\ \ true}\,w}{A \wedge A\ \ true}\ \wedge I}{B\ \ true}\ \supset E}{A \supset B\ \ true}\ \supset I^w \quad \dfrac{\dfrac{}{A \wedge (A \wedge A \supset B)\ true}\,u}{A\ \ true}\ \wedge E_1}{B\ \ true}\ \supset E}{A \wedge (A \wedge A \supset B) \supset B\ \ true}\ \supset I^u$$

By using the rules we have just learned, we can transform it in a proof annotated with terms (in red):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{u : A \wedge (A \wedge A \supset B)}}{\mathbf{snd}\ u : A \wedge A \supset B}\wedge E_2 \quad \dfrac{\dfrac{}{w : A} \quad \dfrac{}{w : A}}{\langle w, w\rangle\ A \wedge A}\wedge I}{\mathbf{snd}\ u\ \langle w, w\rangle :\ B}\supset E}{\lambda w.\mathbf{snd}\ u\ \langle w, w\rangle : A \supset B}\supset I^w \quad \dfrac{\dfrac{}{u : A \wedge (A \wedge A \supset B)}}{\mathbf{fst}\ u : A}\wedge E_1}{(\lambda w.\mathbf{snd}\ u\ \langle w, w\rangle)\mathbf{fst}\ u : B}\supset E}{\lambda u.(\lambda w.\mathbf{snd}\ u\ \langle w, w\rangle)\mathbf{fst}\ u : A \wedge (A \wedge A \supset B) \supset B}\supset I^u$$

The term $\lambda u.(\lambda w.\mathbf{snd}\ u\ \langle w, w\rangle)\mathbf{fst}\ u$ on the conclusion is called the *proof term* of this proof. It can be thought of as an abbreviation of how the proof was constructed. Notice that if someone gives you only this term, you are able to reconstruct exactly this proof. There are other proofs for this same formula, but using this proof term you can only reconstruct this one! That is what *proofs as terms* and *formulas as types* is all about :) A term, or a program of a certain type corresponds to a proof of the formula, and checking a proof then reduces to type checking a program.

If you are familiar with λ calculus or with proofs, you might have noticed some redundancy either on that term or in the proof. On the term level, there is a *redex*, i.e., a part that can be simplified, in this case, by function application. On the proof level, there is an implication introduction constructing $A \supset B$ followed immediately by its elimination, which is redundant. These are exactly the same thing: the redex is caused by the redundancy in the proof and vice versa. If we transform the proof, or the term, we get the new annotated proof below:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{u : A \wedge (A \wedge A \supset B)}}{\mathbf{snd}\ u : A \wedge A \supset B}\wedge E_2 \quad \dfrac{\dfrac{\dfrac{}{u : A \wedge (A \wedge A \supset B)}}{\mathbf{fst}\ u : A}\wedge E_1 \quad \dfrac{\dfrac{}{u : A \wedge (A \wedge A \supset B)}}{\mathbf{fst}\ u : A}\wedge E_1}{\langle \mathbf{fst}\ u, \mathbf{fst}\ u\rangle : A \wedge A}\wedge I}{\mathbf{snd}\ u\ \langle \mathbf{fst}\ u, \mathbf{fst}\ u\rangle : B}\supset E}{\lambda u.\mathbf{snd}\ u\ \langle \mathbf{fst}\ u, \mathbf{fst}\ u\rangle : A \wedge (A \wedge A \supset B) \supset B}\supset I^u$$

Now we have no more redexes and no more ways to simplify the proof. How cool is that? Turns out that normalizing proofs and (typed) λ-terms (or programs) is the same thing. That's Curry-Howard.

3.2 What about a real world function?

The Curry-Howard isomorphism states that one can map programs of type T to proofs of a proposition T. In particular, a program of type $A \supset A$ is

mapped to a function $\lambda x.x$, which is the identity function. At this point one might wonder if all functions of type $A \supset A$ reduce to the identity function. Well, certainly not! Take list sorting for instance. The type of a list sorting function would be `(int list) -> (int list)` and this is definitely not the identity function (unless you live in a parallel world where all lists are sorted). What happened with the isomorphism there?

There are two things going on here. First of all, the isomorphism will work if *no* assumption is made about the types. As soon as you decide that your A is `int list`, you give some structure to the type and use that structure in your program. Curry-Howard does not work this way. In fact, it can be shown that, if no assumptions are made with respect to A (i.e., if it is only a generic type), any function with type $A \supset A$ can be reduced to the identity function. Neat, no?

The second thing is that the isomorphism works only for a very small notion of programs. Think about it. Propositional intuitionistic logic is decidable. That means that, for every formula, we can either find a proof for it or prove that there is no proof. On the programming side this results on terminating programs, since you can decide if that program computes a result of that type or not. So the programming language fragment we can cover with our logic is not Turing complete. Actually, if we were to include (general) recursion in the language, this would result in an inconsistent logic! Suddenly programs can loop forever, and we can derive $\bot$.

In any case, the Curry-Howard correspondence is important because it organizes perspectives. And this was only the first step. After that, people realized that many other logics can account for many other constructions in programming languages, and that is an active research field. This kind of mapping of formalisms is very useful to look at things from a different perspective and to translate results from one side to another (e.g., typed λ-calculus is normalizing, therefore, so are proofs in our logic). Another result of this kind is the Church-Turing thesis, for example, which relates recursive functions and Turing machines.

4 Sequent Calculus

Gentzen's motivation for coming up with natural deduction was to formalize mathematical reasoning so that he could prove consistency of arithmetic. He started by formalizing pure logical reasoning and showing that that system was consistent, and later adding arithmetic. Unfortunately, showing consistency in natural deduction turned out to be much harder than expected[7]. So much so that he decided to invent another calculus in which reasoning would be easier. Hence, sequent calculus was born.

In order to show that the proposed sequent calculus is sound and complete with respect to natural deduction, we will get to it in three steps: first we simply change the notation of natural deduction, then, using this new notation, we show soundness and completeness w.r.t. sequent calculus using some extra inference rules (namely: cut, identity, and weakening). Finally, we could show that removing these extra rules results in an equally powerful calculus[8]. This is depicted in Figure 3, where the symbols denote:

- $\equiv$ The systems are equivalent, they only use different notations and proofs can be mapped back and forth easily.
- $\Updownarrow$ The systems are sound and complete w.r.t. each other, and therefore can be considered equivalent, or equally powerful (we prove this in this section).
- $\updownarrow$ The systems are shown to be equivalent by showing that the rules cut, identity and weakening are *admissible* (i.e., all proofs using these rules can be transformed into proofs that do not use these rules).

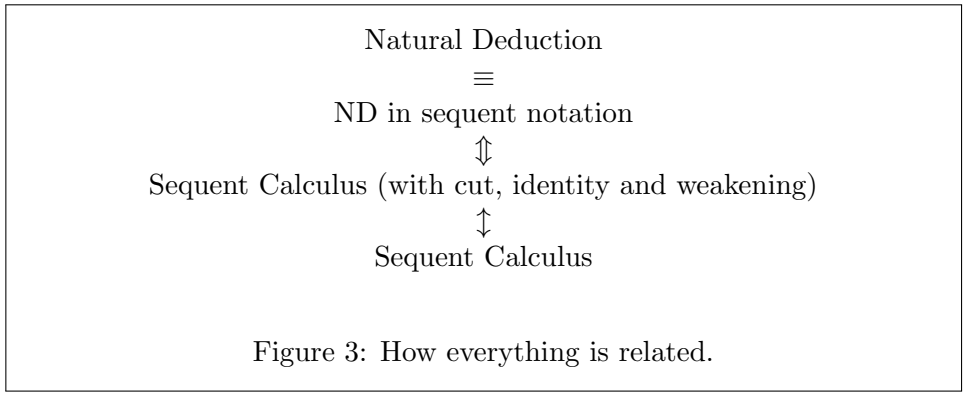

Figure 3: How everything is related.

[7] Proved in Dag Prawitz's thesis: *Natural Deduction: A proof-theoretical study* (1965).
[8] For the sake of time, we will only state these theorems, without proving them.

4.1 Natural Deduction in Sequent Notation

We start with the new notations. The sequent notation for natural deduction will bring the hypotheses available for proving a formula to the same level as the formula, in judgments of the shape:

$$\Gamma \vdash A$$

Where Γ denotes a (possibly empty) (multi-)set of formulas. This will result in more linear and less two-dimensional proofs. The idea is simple: whenever new assumptions are created by a rule application, we will add them to the Γ context as opposed to writing them somewhere above the proof. This will also make the scope clearer. The translation of natural deduction rules to a sequent notation are shown in Figure 4. Note that we have added hypothesis discharge, which is equivalent to using a hypothesis to close a proof. The proofs in sequent notation will always finish with applications of *hyp*.

An example of a proof in this calculus is:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \supset B, A \wedge C \vdash A \supset B}\, hyp \quad \dfrac{\dfrac{}{A \supset B, A \wedge C \vdash A \wedge C}\, hyp}{A \supset B, A \wedge C \vdash A}\, \wedge E_1}{A \supset B, A \wedge C \vdash B}\, \supset E \quad \dfrac{\dfrac{}{A \supset B, A \wedge C \vdash A \wedge C}\, hyp}{A \supset B, A \wedge C \vdash C}\, \wedge E_2}{A \supset B, A \wedge C \vdash B \wedge C}\, \wedge I}{A \supset B \vdash (A \wedge C) \supset (B \wedge C)}\, \supset I}{\vdash (A \supset B) \supset ((A \wedge C) \supset (B \wedge C))}\, \supset I$$

We will also make use of the substitution principle, which says that we can replace the use of an assumption (or hypothesis) by a derivation of it if we have it. The *subst* rule can be added to our calculus without compromising it because it is admissible.

Lemma 1 (Admissibility of *subst*)**.** *The subst rule is admissible.*

This can be proved by replacing the application of *subst* into an application of $\supset I$ followed by $\supset E$.

Moreover, we will use the fact that adding more formulas to the context does not affect provability:

Lemma 2 (Weakening Lemma)**.** *If* $\Gamma \vdash C$ *then* $\Gamma, F \vdash C$.

The proof of this lemma follow by structural induction on the proof tree, and is quite straightforward.

Natural Deduction	**ND in sequent notation**
$\dfrac{A\ true \quad B\ true}{A \wedge B\ true}\ \wedge I$	$\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge I$
$\dfrac{A \wedge B\ true}{A\ true}\ \wedge E_1 \qquad \dfrac{A \wedge B\ true}{B\ true}\ \wedge E_2$	$\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \wedge E_1 \qquad \dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge E_2$
$\dfrac{A\ true}{A \vee B\ true}\ \vee I_1 \qquad \dfrac{B\ true}{A \vee B\ true}\ \vee I_2$	$\dfrac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee I_1 \qquad \dfrac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee I_2$
$\dfrac{A \vee B\ true \quad \begin{array}{c}\overline{A\ true}^{\,u} \\ \vdots \\ C\ true\end{array} \quad \begin{array}{c}\overline{B\ true}^{\,v} \\ \vdots \\ C\ true\end{array}}{C\ true}\ \vee E^{u,v}$	$\dfrac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}\ \vee E$
$\dfrac{\begin{array}{c}\overline{A\ true}^{\,u} \\ \vdots \\ B\ true\end{array}}{A \supset B\ true}\ \supset I^u$	$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}\ \supset I$
$\dfrac{A \supset B\ true \quad A\ true}{B\ true}\ \supset E$	$\dfrac{\Gamma \vdash A \supset B \quad \Gamma \vdash A}{\Gamma \vdash B}\ \supset E$
$\dfrac{}{\top\ true}\ \top I$	$\dfrac{}{\Gamma \vdash \top}\ \top I$
$\dfrac{\bot\ true}{C\ true}\ \bot E$	$\dfrac{\Gamma \vdash \bot}{\Gamma \vdash C}\ \bot E$
Hypothesis discharging	$\dfrac{}{\Gamma, A \vdash A}\ hyp$
$\begin{array}{ccc}\overline{A\ true} & & \overset{\mathcal{D}}{A\ true} \\ \vdots & & \vdots \\ \text{if } C\ true & \text{and} & \text{then } C\ true\end{array}$	$\dfrac{\Gamma, A \vdash C \quad \Gamma \vdash A}{\Gamma \vdash C}\ subst$

Figure 4: Natural Deduction ≡ ND in sequent notation

$$\frac{\Gamma \to A \quad \Gamma \to B}{\Gamma \to A \wedge B}\ \wedge R \qquad \frac{\Gamma, A, B \to C}{\Gamma, A \wedge B \to C}\ \wedge L$$

$$\frac{\Gamma \to A}{\Gamma \to A \vee B}\ \vee R_1 \quad \frac{\Gamma \to B}{\Gamma \to A \vee B}\ \vee R_2 \qquad \frac{\Gamma, A \to C \quad \Gamma, B \to C}{\Gamma, A \vee B \to C}\ \vee L$$

$$\frac{\Gamma, A \to B}{\Gamma \to A \supset B}\ \supset R \qquad \frac{\Gamma, A \supset B \to A \quad \Gamma, B \to C}{\Gamma, A \supset B \to C}\ \supset L$$

$$\frac{}{\Gamma \to \top}\ \top R \qquad \frac{}{\Gamma, \bot \to C}\ \bot L \qquad \frac{}{\Gamma, p \to p}\ \mathit{init} \text{(for atomic } p\text{)}$$

Figure 5: Sequent calculus for intuitionistic logic (LJ)

4.2 Sequent Calculus

The judgments of sequent calculus will be of the shape:

$$\Gamma \to A$$

Where Γ denotes a (possibly empty) (multi-)set of formulas. These are called *sequents*. One should not be fooled by the similarity between this and the last judgment. The main (important) difference is that Γ now holds everything that is *used* in a proof, while the right side is reserved for the formula to be *verified*. This means that we will have rules operating on the left side of the sequent. The sequent calculus rules for intuitionistic logic are shown on Figure 4.2. This calculus is sometimes called LJ. Note that rule names have changed in order to reflect the fact that we are working either on the left or right side of the arrow (as opposed to eliminating and introducing connectives). A nice side effect of the change to sequent calculus is that now proofs are completely linear, and proof-search is performed solely from the bottom up.

As an example, here's a proof of $(A \supset B) \supset ((A \wedge C) \supset (B \wedge C))$ in sequent calculus:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \supset B, A, C \to A}\,init}{A \supset B, A \wedge C \to A}\,\wedge L \quad \dfrac{}{A \wedge C, B \to B}\,init}{A \supset B, A \wedge C \to B}\,\supset L \quad \dfrac{\dfrac{}{A \supset B, A, C \to C}\,init}{A \supset B, A \wedge C \to C}\,\wedge L}{A \supset B, A \wedge C \to B \wedge C}\,\wedge R}{A \supset B \to (A \wedge C) \supset (B \wedge C)}\,\supset R}{\to (A \supset B) \supset ((A \wedge C) \supset (B \wedge C))}\,\supset R$$

In order to show soundness and completeness w.r.t. natural deduction, it will be helpful to have the following rules in addition to the ones in Figure 4.2:

$$\dfrac{}{\Gamma, A \to A}\,id \qquad \dfrac{\Gamma \to A \quad \Gamma, A \to C}{\Gamma \to C}\,cut \qquad \dfrac{\Gamma \to C}{\Gamma, A \to C}\,weak$$

id is basically *init* generalized for arbitrary formulas, *cut* is a sort of substitution lemma[9] and *weak* is weakening (which we proved admissible for ND in sequent notation). Using those rules will be harmless because they are admissible. These proofs are beyond the scope of this tutorial due to time constraints.

4.3 Soundness and Completeness

Right now we will worry ourselves with showing that the sequent calculus (+ id, cut and weakening) is sound and complete with respect to natural deduction. This will be proved by a structural induction on the proof tree, transforming proofs in one formalism into the other. Using the sequent notation for natural deduction will make the proof transformations quite natural (for half the cases at least).

In what follows, we will denote the derivation $\Gamma \overset{\mathcal{D}}{\vdash} A$ by the 1-dimensional notation $\mathcal{D} :: \Gamma \vdash A$. Similarly for $\to$-sequents.

Theorem 2. *(Soundness) If* $\mathcal{D} :: \Gamma \to A$ *then* $\mathcal{D}' :: \Gamma \vdash A$.

Proof. The proof proceeds by showing how to transform a derivation of $\mathcal{D}$ into $\mathcal{D}'$. This is done inductively on the structure of $\mathcal{D}$.

Base cases: The base cases are those where $\mathcal{D}$ is "empty", i.e., $\Gamma \to A$ has no premises. The transformations are as follows:

[9] Fact: cut corresponds to the use of lemmas in mathematical proofs.

$$\frac{}{\Gamma, A \to A}\, id \quad \rightsquigarrow \quad \frac{}{\Gamma, A \vdash A}\, hyp$$

$$\frac{}{\Gamma \to \top}\, \top R \quad \rightsquigarrow \quad \frac{}{\Gamma \vdash \top}\, \top I$$

$$\frac{}{\Gamma, \bot \to C}\, \bot L \quad \rightsquigarrow \quad \frac{\dfrac{}{\Gamma, \bot \vdash \bot}\, hyp}{\Gamma, \bot \vdash C}\, \bot E$$

Inductive cases: Each inductive case corresponds to how $\mathcal{D}$ might be constructed, thus we case on the lowermost rule in the proof. Our inductive hypotheses will state that the theorem holds for the structurally smaller proofs that are above this lowermost rule of $\mathcal{D}$. Let these proofs be called $\mathcal{E}$ and $\mathcal{F}$ (because all rules have, at most, two premises), then:

IH1: If $\mathcal{E} :: \Gamma \to A$ then $\mathcal{E}' :: \Gamma \vdash A$.

IH2: If $\mathcal{F} :: \Gamma \to B$ then $\mathcal{F}' :: \Gamma \vdash B$.

Using the inductive hypotheses, we can define the following transformations from a $\to$-derivation to a $\vdash$-derivation:

Case $\wedge R$:

$$\frac{\begin{array}{cc}\mathcal{E} & \mathcal{F} \\ \Gamma \to A & \Gamma \to B\end{array}}{\Gamma \to A \wedge B}\, \wedge R \quad \rightsquigarrow \quad \frac{\begin{array}{cc}\mathcal{E}' & \mathcal{F}' \\ \Gamma \vdash A & \Gamma \vdash B\end{array}}{\Gamma \vdash A \wedge B}\, \wedge I$$

Case $\wedge L$:

$$\frac{\begin{array}{c}\mathcal{E} \\ \Gamma, A, B \to C\end{array}}{\Gamma, A \wedge B \to C}\, \wedge L$$

$$\Big\downarrow$$

$$\dfrac{\dfrac{\mathcal{E}' + \text{weak. lemma}}{\Gamma, A \wedge B, A, B \vdash C} \quad \dfrac{\dfrac{}{\Gamma, A \wedge B, A \wedge B \vdash A \wedge B}\, hyp}{\Gamma, A \wedge B, A \vdash B}\, \wedge E_2}{\Gamma, A \wedge B, A \vdash C}\, subst \quad \dfrac{\dfrac{}{\Gamma, A \wedge B \vdash A \wedge B}\, hyp}{\Gamma, A \wedge B \vdash A}\, \wedge E_1}{\Gamma, A \wedge B \vdash C}\, subst$$

Case $\vee R_1$: (the case for $\vee R_2$ is analogous)

$$\frac{\begin{array}{c}\mathcal{E} \\ \Gamma \to A\end{array}}{\Gamma \to A \vee B}\, \vee R_1 \quad \rightsquigarrow \quad \frac{\begin{array}{c}\mathcal{E}' \\ \Gamma \vdash A\end{array}}{\Gamma \vdash A \vee B}\, \vee I_1$$

Case $\vee L$:

$$\dfrac{\dfrac{\overset{\mathcal{E}}{\Gamma, A \to C} \quad \overset{\mathcal{F}}{\Gamma, B \to C}}{\Gamma, A \vee B \to C}\ \vee L}{\dfrac{\dfrac{}{\Gamma, A \vee B \vdash A \vee B}\ hyp \quad \begin{array}{c}\mathcal{E}' + \text{weak. lemma} \\ \Gamma, A \vee B, A \vdash C\end{array} \quad \begin{array}{c}\mathcal{F}' + \text{weak. lemma} \\ \Gamma, A \vee B, B \vdash C\end{array}}{\Gamma, A \vee B \vdash C}\ \vee E}$$

Case $\supset R$:

$$\dfrac{\overset{\mathcal{D}}{\Gamma, A \to B}}{\Gamma \to A \supset B}\ \supset R \quad \rightsquigarrow \quad \dfrac{\overset{\mathcal{D}'}{\Gamma, A \vdash B}}{\Gamma \vdash A \supset B}\ \supset I$$

Case $\supset L$:

$$\dfrac{\dfrac{\overset{\mathcal{D}}{\Gamma, A \supset B \to A} \quad \overset{\mathcal{E}}{\Gamma, B \to C}}{\Gamma, A \supset B \to C}\ \supset L}{\dfrac{\begin{array}{c}\mathcal{E}' + \text{weak. lemma} \\ \Gamma, A \supset B, B \vdash C\end{array} \quad \dfrac{\dfrac{}{\Gamma, A \supset B \vdash A \supset B}\ hyp \quad \overset{\mathcal{D}'}{\Gamma, A \supset B \vdash A}}{\Gamma, A \supset B \vdash B}\ \supset E}{\Gamma, A \supset B \vdash C}\ subst}$$

□

Theorem 3. *(Completeness) If $\mathcal{D} :: \Gamma \vdash A$ then $\mathcal{D}' :: \Gamma \to A$.*

Proof. The proof proceeds by structural induction on $\mathcal{D}$.

Base cases:

$$\dfrac{}{\Gamma \vdash \top}\ \top I \quad \rightsquigarrow \quad \dfrac{}{\Gamma \to \top}\ \top R$$

$$\dfrac{}{\Gamma, A \vdash A}\ hyp \quad \rightsquigarrow \quad \dfrac{}{\Gamma, A \to A}\ init$$

Inductive cases: There are nine cases, one for each (non-axiomatic) rule that might be used to construct $\mathcal{D}$. We show only a few of them. In each case, we assume the following inductive hypotheses:

IH1: If $\mathcal{E} :: \Gamma \vdash A$ then $\mathcal{E}' :: \Gamma \to A$.

IH2: If $\mathcal{F} :: \Gamma \vdash B$ then $\mathcal{F}' :: \Gamma \to B$.

Case $\wedge I$:

$$\dfrac{\begin{array}{cc}\mathcal{E} & \mathcal{F}\\ \Gamma\vdash A & \Gamma\vdash B\end{array}}{\Gamma\vdash A\wedge B}\wedge I \quad\rightsquigarrow\quad \dfrac{\begin{array}{cc}\mathcal{E}' & \mathcal{F}'\\ \Gamma\to A & \Gamma\to B\end{array}}{\Gamma\to A\wedge B}\wedge R$$

Case $\wedge E_1$:

$$\dfrac{\begin{array}{c}\mathcal{E}\\ \Gamma\vdash A\wedge B\end{array}}{\Gamma\vdash A}\wedge E_1 \quad\rightsquigarrow\quad \dfrac{\begin{array}{cc}\mathcal{E}' & \dfrac{}{\Gamma, A, B\to A}\,id\\ \Gamma\to A\wedge B & \dfrac{}{\Gamma, A\wedge B\to A}\wedge L\end{array}}{\Gamma\to A}\,cut$$

Case $\vee I_1$: (the case of $\vee I_2$ is analogous)

$$\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma\vdash A\end{array}}{\Gamma\vdash A\vee B}\vee I_1 \quad\rightsquigarrow\quad \dfrac{\begin{array}{c}\mathcal{D}'\\ \Gamma\to A\end{array}}{\Gamma\to A\vee B}\vee R_1$$

Case $\vee E$:

$$\dfrac{\begin{array}{ccc}\mathcal{D} & \mathcal{E} & \mathcal{F}\\ \Gamma\vdash A\vee B & \Gamma, A\vdash C & \Gamma, B\vdash C\end{array}}{\Gamma\vdash C}\vee E \quad\rightsquigarrow\quad \dfrac{\begin{array}{cc}\mathcal{D}' & \dfrac{\begin{array}{cc}\mathcal{E}' & \mathcal{F}'\\ \Gamma, A\to C & \Gamma, B\to C\end{array}}{\Gamma, A\vee B\to C}\vee L\\ \Gamma\to A\vee B & \end{array}}{\Gamma\to C}\,cut$$

Case $\supset I$:

$$\dfrac{\begin{array}{c}\mathcal{E}\\ \Gamma, A\vdash B\end{array}}{\Gamma\vdash A\supset B}\supset I \quad\rightsquigarrow\quad \dfrac{\begin{array}{c}\mathcal{E}'\\ \Gamma, A\to B\end{array}}{\Gamma\to A\supset B}\supset R$$

Case $\supset E$:

$$\dfrac{\begin{array}{cc}\mathcal{E} & \mathcal{F}\\ \Gamma\vdash A\supset B & \Gamma\vdash A\end{array}}{\Gamma\vdash B}\supset E$$

$$\Big\downarrow$$

$$\dfrac{\begin{array}{cc}\mathcal{E}' & \dfrac{\begin{array}{cc}\dfrac{\begin{array}{c}\mathcal{F}'\\ \Gamma\to A\end{array}}{\Gamma, A\supset B\to A}\,weak & \dfrac{}{\Gamma, A\supset B, B\to B}\,id\end{array}}{\Gamma, A\supset B\to B}\supset L\\ \Gamma\to A\supset B & \end{array}}{\Gamma\to B}\,cut$$

Case $\bot E$:

$$\dfrac{\dfrac{\mathcal{E}}{\Gamma \vdash \bot}\,\bot E}{\Gamma \vdash C} \quad \rightsquigarrow \quad \dfrac{\dfrac{\mathcal{E}'}{\Gamma \to \bot} \quad \dfrac{}{\Gamma, \bot \to C}\,\bot L}{\Gamma \to C}\,cut$$

□

4.4 Concluding Remarks

Now that we have shown that sequent calculus + cut + id + weakening is sound and complete with respect to natural deduction, we need to prove that the rules cut, id and weakening are *admissible*. This means that any proof with these rules can be transformed into a proof without them. These proofs of admissibility are done via structural induction on the derivation and/or formulas. Unfortunately, we do not have enough time to go over them in this tutorial.

It is important to understand the consequences of these properties, though.

Consequences of id expansion

Suppose we are implementing a calculus that has *id* as a rule, instead of *init*. This means that, at *any* point in the proof, we can check if the sequent can be closed. This is computationally expensive. Not only we need to try all formulas, we have to compare them for equality, which is done on linear time on the size of the formula. If there are many, and big, formulas, this can slow down the prover considerably. Therefore, depending on the implementation, it may be more efficient to check for axioms only on atoms (which, for first-order logic, can be hard enough already).

The fact that we can get rid of *id* and only use *init* is precisely what is called id-expansion.

Consequences of weakening admissibility

Weakening is one of the so-called *structural* rules, because it only changes the structure of the sequent, without touching the formulas themselves. Another popular structural rule is *contraction*:

$$\dfrac{\Gamma, A, A \to C}{\Gamma, A \to C}\,cont$$

The downside of having structural rules in a calculus that is going to be used for automated theorem proving is that these rules can always be applied. This increases the search space considerably, so proving that the system is equally expressive without these rules is of great help.

Both contraction and weakening are admissible for the sequent calculus we have shown previously.

Consequences of cut-elimination

Sub-formula property :) Without cut, we get the nice property that all sequents occurring in a proof contain only sub-formulas of the end-sequent.

Proof-search :) If we have cut in our calculus, then proof search becomes sort of impossible, as we could "guess" a cut formula at every step, and there are infinitely many possibilities. Therefore, a cut-free calculus is much better suitable for automated proof search.

Consistency proof :) In a cut-free calculus it is straightforward that we cannot derive $\rightarrow \bot$, so we get the calculus' consistency for free as a corollary.

Increase in proof length :(There is no free lunch. Turns out that eliminating the cuts from a proof may increase its size exponentially (in propositional logic) or even non-elementary (in first-order logic).